

C12

Thu Dec 27 11:29:33 2007

Function Index

Page 1

DRIVERSTVC_INIT	5	(EMODDER, RSTAVC_CC)
EMODDERRTTC	14	(EMODDER, RSTAVC_CC)
LockSession	2	(EMODDER, RSTAVC_CC)
UBLICKVIEWTEX	4	(EMODDER, RSTAVC_CC)
emodder_create_dsp_client_connection	12	(EMODDER, RSTAVC_CC)
emodder_send_email_to_private_svc	10	(EMODDER, RSTAVC_CC)
emodder_send_wildcard_private_svc	11	(EMODDER, RSTAVC_CC)

Function Index

Page 2


```

1  /*
2   * =====
3   * Copyright 1995,1997 EMC Corporation
4   * =====
5   */
6  /**
7   * =====
8   * DDSSvc_init.c
9   * =====
10  * Mission Statement:
11  */
12  /**
13  * =====
14  * Primary Data Acted On:
15  */
16  /**
17  * Compiler-Time Options:
18  */
19  /**
20  * =====
21  * Basic idea here:
22  */
23  /**
24  */
25 /**
26  * =====
27  * The following provides an RCS_id in the binary that can be located
28  * with the what() utility. The intent is to keep this short.
29  */
30 /**
31  * RCS_id () = "# $Id: Edmddr.c,v $"
32  * Revision: 1.23 $ "
33  */
34 /**
35  */
36 /**
37  * #define _POSIX_SOURCE
38  * unable to compile with this define set */
39 /**
40 /**
41 /**
42 /**
43 /**
44 /**
45 /**
46 /**
47 /**
48 /**
49 /**
50 /**
51 /**
52 /**
53 /**
54 /**
55 /**
56 /**
57 /**
58 /**
59 /**
60 /**
61 /**
62 /**
63 /**
64 /**
65 /**
66 /**
67 /**
68 /**
69 /**
70 /**
71 /**
72 /**
73 /**
74 /**
75 /**
76 /**
77 /**
78 /**
79 /**
80 /**
81 /**
82 /**
83 /**
84 /**
85 /**
86 /**
87 /**
88 /**
89 /**
90 /**
91 /**
92 /**
93 /**
94 /**
95 /**
96 /**
97 /**
98 /**
99 /**
100 /**
101 /**
102 /**
103 /**
104 /**
105 /**
106 /**
107 /**
108 /**
109 /**
110 /**
111 /**
112 /**
113 /**
114 /**
115 /**
116 /**
117 /**
118 /**
119 /**
120 /**
121 /**
122 /**

```

```

123 1 {
124 1     static boolean_t bFirst = TRUE;
125 1
126 1     if (first == TRUE)
127 1     {
128 2         first = FALSE;
129 2         pthread_mutex_init(&g_servicemtx, NULL);
130 1
131 1     }
132 1     pthread_mutex_lock(&g_servicemtx);
133 1
}

```

```

135 //*****
136 /**
137 ** Routine: unlockSyncMutex
138 ** Inputs: None
139 ** Outputs: None
140 ** Return Codes: None
141 ** Purpose: Unlock the mutex for service execution
142 */
143 /**
144 *//**
145 */
146 static void
147 unlockSyncMutex()
148 {
149     pthread_mutex_unlock(&g_servicemtx);
150 }

```


UDRS_SVC_Init Page 9 of 16

Thu Dec 27 11:29:33 2007

Page 10 of 16

```

398 1 // Insert handle object into Global list.
399 1 // Insert handle object into Global list.
400 1
401 1 // lrc = newHandleSet( &sID,
402 1 //   fcl,
403 1 //   fcl2,
404 1 //   &shellHandle,
405 1 //   &status );
406 1
407 1 if ( 0 != lrc )
408 1 {
409 1     (void) free(svc_rpc_h);
410 2     _EDB_DISPATCH_LOGENT(
411 2         _FILE_, LOG_ERR, DPD_HANDLE_INSERTION_ERROR,
412 2         status, "newHandleSet() failure");
413 2     P_dso->setStatistic(ID_SERVICE_FAILURE_NONEXC);
414 2     pthread_exit( NULL );
415 1 }
416 1
417 1 // Let's clean up and set the status to RUNNING.
418 1 // D_p-> setStatus( ID_SERVICE_RUNNING );
419 1
420 1 // unlockService();
421 1
422 1 // pthread_exit( NULL );
423 1
424 1 return NULL;
425 1
426 1
427 1
428 1 // Function:
429 1 // Description:
430 1 //   -0 Success
431 1 //   +1 Read Failure
432 1 //   +2 Write Failure
433 1 //   +3 Read Less than expected
434 1 //   +4 Write More than expected
435 1 //   +5 Invalid handle
436 1 //   +6 Invalid parameter
437 1 //   +7 Invalid file handle
438 1 //   +8 Invalid pipe handle
439 1 //   +9 Invalid connection handle
440 1 //   +A Invalid service handle
441 1 auto int lrc=0;
442 1 auto unsigned char *p_client_hndl;
443 1
444 1 // Isolate the connection handle from the server 'if_spec'.
445 1 // The IP/PORT are part of the created if_spec structure.
446 1 // The IP/PORT are part of the created if_spec structure.
447 1 // P_Client_h = DispatchClient->spec.connect_handle_D;
448 1
449 1 // Client_h = DispatchClient->spec.connect_handle_D;
450 1 // Write the handle to the service so it can contact me
451 1 // Write the handle to the service so it can contact me
452 1

```

```

434 1         P_CLIENT_H;
435 1         CONNECT_HANDLE_SIZE);
436 1     if (CONNECT_HANDLE_SIZE != lrc)
437 1     {
438 1         (void) free(pClient->h);
439 2         END_DISPATCH_LOGIC(LINE,...,LOG_ERR,DOS_WRITE_CHANNEL,...);
440 2     }
441 1     return(-1);
442 1 }
443 1
444 1 return(0);
445 1

```

```

467 /**
468 ** =====
469 ** Function: edmst_send_udt_to_private_svc()
470 ** =====
471 ** Description:
472 ** =====
473 ** Returns:
474 **   0 Successful
475 **   -1 Read Failure
476 **   <0 Read less than expected
477 ** =====
478 */
479 int
480 edmst_send_udt_to_private_svc(int piperovc,
481                                PDEMSession *pSessionObj)
482 {
483     int lrc=0;
484     auto ID_CLIENT_SESSION_ID uid;
485
486     // Write the handle to the service so it can contact me
487     if (pSessionObj->getSessionID(uid))
488     {
489         PDEMSessionObj = cdmst_MrChannel((void*)uid);
490         uid = cdmst_getClientSessionID();
491         if ((sizeof(ID_CLIENT_SESSION_ID)) != lrc)
492             return(-1);
493
494         if ((EMDspatch_Logent(FILE_, LINE_, LOG_ERR,DNP_WRITE_CHANNEL,
495                               0, "edmst_WrChannel() failure"));
496             return(-1);
497     }
498
499     return(0);
500 }

```

```

501 /**
502 ** =====
503 ** Function: edmst_create_dcp_client_connection()
504 ** =====
505 ** Description:
506 ** =====
507 ** Returns:
508 **   0 Successful
509 **   -1 Read Failure
510 **   <0 Read less than expected
511 ** =====
512
513 */
514 int
515 edmst_create_dcp_client_connection(int piperovc,
516                                   RPC_BINDING_HANDLE_T **pbc,
517                                   PDEMSession *p_So)
518 {
519     int lrc;
520     unsigned char *p_restore_service=NULL;
521     error_status_t status;
522     rpc_if_handle_t *pvc_ifspec=NULL;
523
524     // We now need to get the details from the restore service on
525     // how to connect from the dispatch demon now to this restore
526     // service ccc. At this point, the restore service will be send -
527     // ing the restore service core handle information. Then we'll ip
528     // aze the key information needed to create the dcp core handle.
529
530     lrc = edmst_get_client_handle(piperovc, SP_RESTORE_SERVICE );
531
532     if ( lrc != lrc )
533     {
534         EMDspatch_Logent(FILE_, LINE_, LOG_ERR,DNP_GET_CLIENT_HANDLE,
535                           0, "client get client handle() failure");
536
537         p_So->setStatus(DP_SERVICE_FAILURE_NOMEM);
538
539         return(-1);
540     }
541
542     // Create an ifspec from the handle
543     pvc_ifspec = (rpc_if_handle_t *)calloc(1,sizeof(rpc_if_handle_t));
544
545     if (pvc_ifspec == NULL)
546     {
547         EMDspatch_Logent(FILE_, LINE_, LOG_ERR,DNP_NO_MEMORY,
548                           0, "ifspec calloc() failure");
549
550         return(-1);
551     }
552
553     lrc = csc_private_ifspec_init( p_restore_service,
554                                   EDM_DISPATCH_PROTOCOL_CLIENT,
555                                   EDM_DISPATCH_FUNCTIONS,
556                                   &status );
557
558     if ( lrc != lrc )
559     {
560         EMDspatch_Logent(FILE_, LINE_, LOG_ERR,DNP_IFSPEC_INIT_FAILURE,
561                           0, "csc_private_ifspec init() failure");
562
563         return(-1);
564     }
565 }

```

```

566 1 // C:\dev\bus\On()
567 2
568 1 ENDI_DISPATCH_LOGEST( _FILE____LINE____LOG_INFO_ONE_PORT_NUMBERS,
569 2 0, "PORT_INNO_P_DPSVC; i_spec(DECIM) port: %d",
570 2 p_dpsvc_ifspec->portnum );
571 1 )
572
573 1 psvc_h = (PDP_BINDING_HANDLE_T *) call1( sizeof(
574 1 // PDP_BINDING_HANDLE_T ); // Using the connect handle (128 bytes) received from the restore
575 1 // service, connect to the restore service.
576 1 // Service, connect to the restore service.
577 1 // NULL;
578 1 lrc = csc_connect_to_async_rpc_service(
579 1 *p_dpsvc_ifspec,
580 1 psvc_h,
581 1 &status );
582 1
583 1 if ( !lrc )
584 2 {
585 1     (void) free(p_dpsvc_ifspec);
586 2     EMDN_DISPATCH_LOGEST( _FILE____LINE____LOG_ERR_DPP_PRIVATE_SVC_CONNECT_FAILURE,
587 1     EMDN_DISPATCH_LOGEST( _FILE____LINE____LOG_ERR_DPP_PRIVATE_SVC_CONNECT_FAILURE,
588 2     _FILE____LINE____LOG_ERR_DPP_PRIVATE_SVC_CONNECT_FAILURE,
589 1     "csc_connect_to_async_rpc_service( %d ) Failure. Status is %d", status );
590 2 }
591 1
592 1 return(0);
593 1
594 1
595 1
}

```

```

658 1 // Register the callback functions.
659 1 //
660 1 lrc = ccc.register_async_server_interface(
661 1     // dispatch callback,
662 1     // dispatch_protocol_service_table,
663 1     // dispatch_protocol_service_protocol,
664 1     // edm_dispatch_protocol_service_protocol,
665 1     // acnc_status );
666 1
667 1 if ( TRUE != lrc )
668 2 {
669 2     EDMDDerLogOpen(
670 2         FILE_LOG_ERR,
671 2         "Failed to register asynchronous server interface.");
672 2
673 1 }
674 1
675 1 }

return 0;

```

Do_spharmen_ccr	14	(EINODI_CCR_CCI)
Do_spharmen_cew	1	(EINODI_CEW_CCI)
SendDoverzRequestMessage	2	(EINODI_CEW_CCI)
SendDoverzRequestMessage	5	(EINODI_CEW_CCI)
SendDoverzRequestMessage	6	(EINODI_CEW_CCI)
SendDoverzRequestMessage	7	(EINODI_CEW_CCI)
SendDoverzRequestMessage	7	(EINODI_CEW_CCI)
SendConnectControlMessage	4	(EINODI_CEW_CCI)
SendP1StatcomTrmsage	9	(EINODI_CEW_CCI)
SendP1InquiryMessage	8	(EINODI_CEW_CCI)


```

3   */
4   ** Copyright 1996-1997 EMC Corporation
5   */
6   /**
7   ** =====
8   ** EMDCC, CEW-C
9   **
10  ** Mission Statement: This is the entry point for the Control Channel.
11  **
12  ** It's main purpose is to write notifications or
13  ** progress to the
14  ** Dispatch Daemon.
15  ** Primary Data Acted On:
16  ** Compile-Time Options:
17  ** USE_SNMPC - Compile source with snmpc support. If
18  ** not set, assume pcp support.
19  ** Basic Idea here: Module for Control Channel writer thread.
20  ** =====
21  ** =====
22  ** =====
23  ** =====
24  ** =====
25  ** =====
26  ** The following provides an RCG id in the binary that can be located
27  ** with the what(1) utility. The intent is to keep this short.
28  ** =====
29  *'#ifndef defined(RCG_ID)
30  *'#define RCG_ID "0x01$RCGID: EMDCC,C,v $"
31  static char RCG_ID [ ] = {0};
32  *'#endif
33  *'$Date: 1997/02/06 20:49:15 $
34  *'$Id: emdcc.c,v 1.1 1997/02/06 20:49:15 $
35  /**
36  ** RCGID - POSTX_SOURCE unable to compile with this define set */
37  ** =====
38  *'#include <esl/r-portable.h>
39  *'#include <esl/r-portable.h>
40  *'#include <esl/r-xport.h>
41  *'#include <esl/r-malloc.h>
42  *'#include <pthread.h>
43  *'#include <esl/esl.h>
44  *'#include <esl/esl-mm.h>
45  *'#include <esl/r/collect.h>
46  *'#include <esl/r/cplusplus.h>
47  *'#endif
48  extern "C" {
49  *'#endif
50  *'$Id: emdcc.c,v 1.1 1997/02/06 20:49:15 $
51  *'$Id: emdcc.c,v 1.1 1997/02/06 20:49:15 $
52  *'#include <esl/r/patch_protocol.h>
53  *'#include <esl/r/patch_protocol_client.h>
54  *'#include <esl/r/patch_protocol_client.h>
55  *'$Id: emdcc.c,v 1.1 1997/02/06 20:49:15 $
56  */

```

```
DipDaemon_Cow Thu Dec 27 11:30:02 2007
DIP_DROP_MESSAGE( 0, "Session <sid> failed to start - drop message.", sid_high, sid_low );
}
continues;
)
/* execute the callback that will process this message */
switch(ResponseMessage)
{
    case Dp_connect_Confirm:
        rc = SendConnectConfirmMessage(
            sid, client_h_d);
        break;
    case Dp_abort_Request:
        rc = SendAbortRequestMessage(
            sid, client_h_d);
        break;
    case Dp_close_request:
        rc = SendCloseRequestMessage(
            sid, client_h_d);
        break;
        /* Dp_close_request:
           break; */
    case Dp_sendPingRequest:
        rc = SendPingRequestMessage(sid, client_h_d);
        break;
    case Dp_event_confirm:
        /* No confirm needed for this message
           break; */
    case Dp_program_confirm:
        /* No confirm needed for this message
           break; */
    case Dp_final_stats_confirm:
        rc = SendFinalStatsConfirmMessage(
            &sid, client_h_d);
        break;
    default:
        EDMDispatchLogent(
            _FILE_, _LINE_, LOG_ERR, DIP_INVALID_MESSAGE,
            "0,"Invalid message type received.");
}
}
/* Check for a shutdown setting */
/* End of forever loop */
}
/* End of DipDaemon_Cow() */

```

```
Page 4 of 28 SendConnectConfirmMessage Thu Dec 27 11:30:02 2007
160 // Function: SendConnectConfirmMessage()
161 // Description: send the confirm connect message to the
162 // dispatch daemon.
163 // static int
164 // SendConnectConfirmMessage(
165 //     DIP_CLIENT_SESSION_ID *sid, RPC_BINDING_HANDLE_T *cint_p)
166 // {
167 //     int *rc = NULL;
168 //     int lrc = 0;
169 //     int savedlrc = 0;
170 //     int status = 0;
171 //     int scstatus = 0;
172 //     if (cint_p == NULL)
173 //     {
174 //         msg_p = (DIP_CONNECT_CONFIRM_MSG*)callout_id.siedc->connect_confirm_msg;
175 //         msg_p->sid_high = sid->high;
176 //         msg_p->sid_low = sid->low;
177 //         rc = Dp_connect_confirm_1(msg_p,&int_p);
178 //         if (lrc != 0)
179 //             (void) EDMDispatchLogent(_FILE_, _LINE_, LOG_INFO,
180 //                                     "DIP_SENDING_MESSAGE_0");
181 //         else
182 //             (void) EDMDispatchLogent(_FILE_, _LINE_, LOG_INFO,
183 //                                     "Sending Dp_connect_confirm_1 message");
184 //     }
185 //     else
186 //     {
187 //         if (lrc != 0)
188 //             (void) EDMDispatchLogent(_FILE_, _LINE_, LOG_INFO,
189 //                                     "DIP_SENDING_MESSAGE_1");
190 //         else
191 //             (void) EDMDispatchLogent(_FILE_, _LINE_, LOG_INFO,
192 //                                     "Sending Dp_connect_confirm_0");
193 //     }
194 //     else
195 //     {
196 //         /* Get the circ_binding handle associated with this sid */
197 //         lrc = GetCircHandle(sid, circ_binding_handle_p,
198 //                             &status);
199 //         if (lrc != 0)
200 //             (void) EDMDispatchLogent(_FILE_, _LINE_, LOG_ERR, DIP_GRC_CSC_HANDLE_FAILURE, status,
201 //                                     "GetCircHandle failed.");
202 //         savedlrc = lrc;
203 //     }
204 //     if (lrc == 1)
205 //     {
206 //         /* Push message to send onto the queue */
207 //         lrc = PushResponseMessage((int) Dp_connect_confirm,
208 //                                 *sid,
209 //                                 client_handle_p,
210 //                                 &status);
211 //     }
212 //     if (lrc == 1)
213 //     {
214 //         EDMDispatchLogent(
215 //             _FILE_, _LINE_, "DIP_ERR, DIP_PUT_RESPONSE_FAILURE, status",
216 //             "PushResponseMessage failed.");
217 //     }
218 //     savedlrc = lrc;
219 // }
}

```

```

    lrc = savedLrc;
221 2
    return lrc;
223 2
}
224 1
)
226 1
return(0);
227 }

// Function: SendAbortRequestMessage()
229 // Description: Send a abort request to a restore service.
230 //               Send a abort request to a restore service.
231 //               Send a abort request to a restore service.
232 //               Send a abort request to a restore service.
233 static int
234 sendAbortRequestMessage(
235     ID_CLIENT_SESSION_ID *ssid, RPC_BINDING_HANDLE_T *cint_p,
236     int *rc);
237
238 ID_ABORT_REQUEST_MSG *msg_p=NULL;
239
240 ID_ABORT_REQUEST_MSG *msg_p=NULL;
241
242 calc1, sizef(DP_ABORT_REQUEST_MSG);
243 msg_p->sid_high = ssid->high;
244 msg_p->sid_low = ssid->low;
245 rc = dp_abort_request_1(msg_p, *cint_p);
246
247 if (isDebug())
248 {
249     (void)EWN_SwitchLogent(FILE_LINE_INFO,
250     "DP_SENDING_ABORT_MESSAGE, 0, 1 message.");
251 }
252
253 free( msg_p );
254
255 return(0);
256
}

```

SendCloseRequestMessage

Page 8 of 28 SendPingRequestMessage

Thu Dec 27 11:30:02 2007

```

257 // Function: SendCloseRequestMessage()
258 // Description:
259 //   Send a close request to a restore service.
260 //   Send a ping request to a restore service.
261 static int
262 sendCloseRequestMessage(
263     DD_CLIENT_SESSION_ID *ssid,
264     DD_BINDING_HANDLE_T *cInt_D )
265 {
266     int *rc;
267
268     DE_CLOSE_REQUEST_MSG *msg_p=NULL;
269
270     msg_p = (DE_CLOSE_REQUEST_MSG *)
271         malloc(sizeof(DE_CLOSE_REQUEST_MSG));
272
273     msg_p->ssid_high = ssid->high;
274     msg_p->ssid_low = ssid->low;
275
276     rc = dp_close_request_l1(msg_p, *cInt_D );
277
278     if (ISDebugOn())
279     {
280         (void) EMDISPATCH_LOGENT( _FILE, _LINE, _LOG_INFO,
281             "Sending dp_close_request_l1 message. ");
282     }
283
284     free( msg_p );
285
286     return(0);
287
288 }
289
290 static int
291 sendPingRequestMessage(
292     DD_CLIENT_SESSION_ID *ssid,
293     DD_BINDING_HANDLE_T *cInt_D )
294 {
295     int *rc;
296
297     DE_PING_REQUEST_MSG *msg_p=NULL;
298
299     msg_p = (DE_PING_REQUEST_MSG *)
300         malloc(sizeof(DE_PING_REQUEST_MSG));
301
302     msg_p->ssid_high = ssid->high;
303     msg_p->ssid_low = ssid->low;
304
305     rc = dp_ping_request_l1(msg_p, *cInt_D );
306
307     if (ISDebugOn())
308     {
309         (void) EMDISPATCH_LOGENT( _FILE, _LINE, _LOG_INFO,
310             "Sending dp_ping_request_l1 message. ");
311     }
312
313     free( msg_p );
314
315     return(0);
316
317 }
```

Page 7 of 23 EDMDD .ccw.cc 7 Thu Dec 27 11:30:02 2007

Page 8 of 28

EDMDD_CCW.CC 8

3

Thu Dec 27 11:30:02 2001

```

113 // Function: SendFinalStateConfirmMessage()
114 // Description: Send a ping request to a restore service.
115 // Parameters: None
116 // Returns: None
117 // Static int
118 // EndOfSessionConfirmMessage()
119 {
120     // DO_CLIENT_SESSION_ID *ssid, XPC_BINDING_HANDLE_T *cInt_D )
121     // Description: Send a ping request to a restore service.
122     // Parameters: None
123     // Returns: None
124     // XPC_BINDING_HANDLE_T *ptr;
125     // DP_FINAL_STATE_CONFIRM_MSG *msg_D=NULL;
126
127     if (cInt_D != NULL)
128     {
129         msg_D = (DP_FINAL_STATE_CONFIRM_MSG*)
130             calloc(1, sizeof(DP_FINAL_STATE_CONFIRM_MSG));
131
132         msg_D->sid = high;
133         msg_D->status = -1;
134         msg_D->action = ssid->key;
135         rc = dp_final_state_confirm_L(msg_D, *cInt_D);
136
137         if (*IsDebugEnabled())
138             DDP_SENDING_MESSAGE(0, "confirm_l",
139             msg_D->sid, msg_D->status);
140
141         free( msg_D );
142
143     }
144
145     ret = removeSession(ssid, &status);
146
147     if (ret == -1)
148     {
149         (void) EMDISPATCH_LOGENT(FILE__LINE__, LOG_ERR,
150             DDP_REMOVE_SESSION_FAILURE, status,
151             "Failure removing session instance from list.");
152     }
153
154     if (ret == -1)
155     {
156         (void) EMDISPATCH_LOGENT(FILE__LINE__, LOG_ERR,
157             DDP_REMOVE_SESSION_FAILURE, status,
158             "Failure getting session handles from list.");
159     }
160
161     if (out != -1 && err != -1)
162     {
163         closeOut();
164         closeErr();
165     }
166
167     ret = deleteHandleSet(ssid, &FLINKHandle, &status);
168
169     if (ret == -1)
170     {
171         (void) EMDISPATCH_LOGENT(FILE__LINE__, LOG_ERR,
172             DDP_DELETE_HANDLE_SET_FAILURE, status,
173             "Failure removing session handles from list.");
174
175     }

```



```

 5 /**
 6 ** =====
 7 ** Copyright 1996,1997 EMC Corporation
 8 ** =====
 9 ** =====
10 ** Mission Statement: This is the entry point for the Control Channel
11 ** thread. Its main purpose is to read asynchronous
12 ** messages from the Dispatcher daemon.
13 ** =====
14 ** Primary Data Acted On:
15 ** =====
16 ** =====
17 ** Compile-Time Options:
18 ** =====
19 ** USE_SUNRPC - Compile source with sunrpc
20 ** not set, assume DCE support. If
21 ** Basic idea here: Module for Control Channel Reader thread.
22 ** =====
23 ** =====
24 ** =====
25 ** =====
26 ** =====
27 ** The following provides an RCS Id in the binary that can be located
28 ** with the what() utility. The intent is to keep this short.
29 ** =====
30 ** =====
31 ** =====
32 ** =====
33 ** =====
34 ** =====
35 ** =====
36 ** =====
37 ** =====
38 ** =====
39 ** =====
40 ** =====
41 ** =====
42 ** =====
43 ** =====
44 ** =====
45 ** =====
46 ** =====
47 ** =====
48 ** =====
49 ** =====
50 ** =====
51 ** =====
52 ** =====
53 ** =====
54 ** =====
55 ** =====
56 ** =====
57 ** =====
58 ** =====
59 ** =====
60 ** =====
61 ** =====
62 ** =====
63 ** =====
64 ** =====
65 ** =====
66 1 // Wait for transient thread to tell me there is something to listen on.
67 1 // -----
68 1 pthread_mutex_lock( &sectorRoy_mutex );
69 1 pthead_cond_wait( &sectorRoy_cv, &sectorRoy_mutex );
70 1 pthread_mutex_unlock( &sectorRoy_mutex );
71 1
72 1
73 1 /**
74 1 * =====
75 1 ** =====
76 1 ** =====
77 1 /**
78 1 {
79 1     lrc = csc_async_server_listen( rcsl_timeout, &status );
80 1
81 1     if ( lrc == lrc )
82 3     {
83 3         EMCDispatch_Logout(
84 3             FILE__, LINE__, LOG_INFO, DDP_FAILED_LISTEN,
85 3             0, "Bad returned from listen.");
86 3     }
87 3
88 3     if ( l == lrc )
89 3     {
90 3         lrc = csc_async_server_listen(
91 3             FILE__, LINE__, LOG_INFO, DDP_LISTEN_TIMEOUT,
92 3             0, "listen() timedout.");
93 3     }
94 1
95 1 /**
96 1 ** =====
97 1 ** Unregister our service upon exit request.
98 1
99 1 /**
100 1 rcs = csc_unregister_async_server_interface( lspec, &status );
101 1 EMCDispatch_Logout( FILE__, LINE__, LOG_INFO, DDP_UNREGISTER_SVC,
102 1
102 1 return( (void*)0 );
102 1

```



```

1  /*
2   ** Copyright 1996, 1997 EMC Corporation
3   */
4
5  /* EDMODHandle.cc
6
7  * Mission Statement: file that contains the Handle class methods
8  */
9
10 /* Primary Data Acted On:
11 */
12 /* Compiler-Time Options:
13 */
14 /* Basic idea here: The Handle object is a container which holds a
15 * set of handles for each running service.
16 */
17 /* */
18 */
19
20 #if !defined(INT)
21     static char RCS_Id [] = "#$Id:$";
22     static const char Revision[] = "Revision: 1.9 $";
23     static const char Date[] = "Date: 1997/02/06 20:49:15 $";
24
25 #endif
26
27 #include <esl/c-portable.h>
28 #include <esl/epnopen.h>;
29 #include <esl/lnout.h>;
30
31 * include <string.h>
32 * include <stlib.h>
33 /* Rogue Wave includes
34 #include <rw/collset.h>
35 #include <rw/vtstream.h>
36 #include <rw/vtstream.h>
37
38 #include <esc/escmm.h>
39
40 #include <eslmln/adminlink/api.h>;
41 #include <escmm/escmmobject.h>;
42 #include <escmm/dispacth.daemon.h>
43 #include <EDMODHandle.h>;
44
45 // Needed for rogue wave linked list manager.
46 // 413 is the object ID.
47 RODLIBN(COLLECTABLE(EDMODHandle, EDMODHANDLE))
48
49 */
50
51 /* Routine: EDMODHandle constructor
52 */
53 /* Inputs: None
54 */
55 /* Outputs: None
56 */
57 /* Return Codes:
58 */
59 /* Purpose: None
60 */
61 /* Purpose: Initializes the Handle class by resetting the internal data.
62 */
63 */
64 */
65 EDMODHandle::EDMODHandle()
66 {
67     rpEBD = NULL;
68     strAID = NULL;
69     strBID = NULL;
70     strCIP = 0;
71 }
72
73 mainst(&sessionID, 0, sizeof(sessionID));
74
75 */
76
77 /* Routine: EDMODHandle constructor
78 */
79 /* Inputs:
80 */
81     int stdcrtipce - the stdio dispatcher of the service
82     int stdcrtrpc - the stdcrt descriptor of the service
83     int stdcrtsession_id *sess - the session ID of the
84     service
85     */
86     /* Outputs: None
87     */
88     /* Return Codes:
89 */
90     /* Purpose: initializes the internals of the Handle class.
91 */
92
93 */
94
95 EDMODHandle::EDMODHandle(
96     IN TPCBindingHandle_t *bh, IN DD_CLIENT_SESSION_ID *sess,
97     IN int stdcrtipce,
98     IN int stdcrtrpc,
99     IN int stdcrtsession_id);
100
101     if (sess != NULL)
102         memcpy(&sessionID, sess, sizeof(DD_CLIENT_SESSION_ID));
103
104 */
105
106 /* Routine: EDMODHandle destructor
107 */
108 /* Inputs: None
109 */
110 /* Outputs: None
111 */
112 /* Return Codes:
113 */
114 /* Purpose: Doesn't really do anything but seems to be a requirement
115 */
116 /* Purpose: for the linked list manager.
117 */
118 */
119 */
120 */
121 */
122 EDMODHandle::~EDMODHandle()

```


22 of 2

```

247    **
248    ** Purpose: Save class data to a stream.
249    **
250    ****
251    ****
252    */
253 void
254 EDMHandle::saveGuts( IN RWostream &strm)
255 {
256 1   // Save parent class data too
257 1   RMCollectable::savemem( strm);
258 1
259 1   // Self as an example
260 1
261 }
262 ****
263 //*****
264 /**
265  ** Routine: restoreGuts
266  ** Inputs: RWfile f - file to read internal data from.
267  ** Outputs: None
268  ** Return Codes:
269  ** Return: None
270  */
271 /**
272  ** Purpose: Restores an instance of the Handle class by reading the data
273  ** from the passed in file.
274  */
275 ****
276 ****
277 ****
278 /**
279  */
280 void
281 EDMHandle::restorguts( IN RWfile &f)
282 1
283 1   // Restore parent data too
284 1   RMCollectable::restoreGuts(f);
285 1
286 1   // Left as an example
287 1
288 /**
289  */
290 /**
291  ** Routine: restoreGuts
292  */
293 /**
294  ** Inputs: RWistream strm - stream to read internal data from.
295  ** Outputs: None
296  */
297 /**
298  ** Return Codes:
299  */
300 /**
301  ** Purpose: Restores an instance of the Handle class by reading the data
302  ** from the passed in stream.
303  */
304 /**
305  */
306 void
307 EDMHandle::restoreGuts( IN RWistream &strm)
308 1
309 1   // Restore parent data too
310 1   RMCollectable::restoreGuts(strm);
311 1
312 1   // Left as an example
313 */
314 /**
315  */
316 /**
317  ** Routine: binaryStoreSize
318  */
319 /**
320  ** Inputs: None
321  ** Outputs: None
322  */
323 /**
324  ** Return Codes: RWspace count - File size of class written to disk in bytes
325  */
326 /**
327  ** Purpose: Returns the size of class if it were stored on disk.
328  */
329 /**
330  */
331 /**
332  ** Purpose: binaryStoreSize() const
333  */
334 /**
335 1   {
336 1     RMCollectable::binaryStoreSize();
337 1     return count;
338 1   }
339 /**
340  ** Routine: getSessionID
341  */
342 /**
343  ** Inputs: None
344  ** Outputs: None
345  */
346 /**
347  ** Return Codes: DD::client::session_id sessionID - the session ID
348  */
349 /**
350  ** Purpose: Returns the ID of the session the object belongs to.
351  */
352 /**
353  ** Inputs: DD::client::session_id sessionId
354  */
355 /**
356  ** Purpose: Returns the ID of the session the object belongs to.
357  */
358 /**
359  */
360 /**
361  ** Routine: setSessionID
362  */
363 /**

```

```

435 /**
436 **          this object
437 */
438 void EmdHandle::setBindingHandle(rpc_binding_handle_t *bh)
439 {
440     EmdHandle::setBindingHandle(bh);
441 }
442 /**
443 **          None
444 */
445 void EmdHandle::setShellHandle(rpc_shellHandle_t *bh)
446 {
447     EmdHandle::setShellHandle(bh);
448 }
449 /**
450 **          None
451 */
452 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
453 {
454     EmdHandle::setSessionID(id);
455 }
456 /**
457 **          None
458 */
459 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
460 {
461     EmdHandle::getBindingHandle(ID);
462 }
463 /**
464 **          None
465 */
466 void EmdHandle::getShellHandle()
467 {
468 }
469 /**
470 **          None
471 */
472 void EmdHandle::setShellHandle(EMD_CLIENT_SESSION_ID id)
473 {
474     EmdHandle::setShellHandle(id);
475 }
476 /**
477 **          None
478 */
479 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID ID)
480 {
481     EmdHandle::setSessionID(ID);
482 }
483 /**
484 **          None
485 */
486 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
487 {
488     EmdHandle::getBindingHandle(ID);
489 }
490 /**
491 **          None
492 */
493 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
494 {
495     EmdHandle::getShellHandle(id);
496 }
497 /**
498 **          None
499 */
500 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
501 {
502     EmdHandle::setSessionID(id);
503 }
504 /**
505 **          None
506 */
507 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
508 {
509     EmdHandle::getBindingHandle(ID);
510 }
511 /**
512 **          None
513 */
514 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
515 {
516     EmdHandle::getShellHandle(id);
517 }
518 /**
519 **          None
520 */
521 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
522 {
523     EmdHandle::setSessionID(id);
524 }
525 /**
526 **          None
527 */
528 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
529 {
530     EmdHandle::getBindingHandle(ID);
531 }
532 /**
533 **          None
534 */
535 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
536 {
537     EmdHandle::getShellHandle(id);
538 }
539 /**
540 **          None
541 */
542 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
543 {
544     EmdHandle::setSessionID(id);
545 }
546 /**
547 **          None
548 */
549 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
550 {
551     EmdHandle::getBindingHandle(ID);
552 }
553 /**
554 **          None
555 */
556 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
557 {
558     EmdHandle::getShellHandle(id);
559 }
560 /**
561 **          None
562 */
563 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
564 {
565     EmdHandle::setSessionID(id);
566 }
567 /**
568 **          None
569 */
570 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
571 {
572     EmdHandle::getBindingHandle(ID);
573 }
574 /**
575 **          None
576 */
577 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
578 {
579     EmdHandle::getShellHandle(id);
580 }
581 /**
582 **          None
583 */
584 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
585 {
586     EmdHandle::setSessionID(id);
587 }
588 /**
589 **          None
590 */
591 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
592 {
593     EmdHandle::getBindingHandle(ID);
594 }
595 /**
596 **          None
597 */
598 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
599 {
600     EmdHandle::getShellHandle(id);
601 }
602 /**
603 **          None
604 */
605 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
606 {
607     EmdHandle::setSessionID(id);
608 }
609 /**
610 **          None
611 */
612 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
613 {
614     EmdHandle::getBindingHandle(ID);
615 }
616 /**
617 **          None
618 */
619 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
620 {
621     EmdHandle::getShellHandle(id);
622 }
623 /**
624 **          None
625 */
626 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
627 {
628     EmdHandle::setSessionID(id);
629 }
630 /**
631 **          None
632 */
633 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
634 {
635     EmdHandle::getBindingHandle(ID);
636 }
637 /**
638 **          None
639 */
640 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
641 {
642     EmdHandle::getShellHandle(id);
643 }
644 /**
645 **          None
646 */
647 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
648 {
649     EmdHandle::setSessionID(id);
650 }
651 /**
652 **          None
653 */
654 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
655 {
656     EmdHandle::getBindingHandle(ID);
657 }
658 /**
659 **          None
660 */
661 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
662 {
663     EmdHandle::getShellHandle(id);
664 }
665 /**
666 **          None
667 */
668 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
669 {
670     EmdHandle::setSessionID(id);
671 }
672 /**
673 **          None
674 */
675 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
676 {
677     EmdHandle::getBindingHandle(ID);
678 }
679 /**
680 **          None
681 */
682 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
683 {
684     EmdHandle::getShellHandle(id);
685 }
686 /**
687 **          None
688 */
689 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
690 {
691     EmdHandle::setSessionID(id);
692 }
693 /**
694 **          None
695 */
696 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
697 {
698     EmdHandle::getBindingHandle(ID);
699 }
700 /**
701 **          None
702 */
703 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
704 {
705     EmdHandle::getShellHandle(id);
706 }
707 /**
708 **          None
709 */
710 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
711 {
712     EmdHandle::setSessionID(id);
713 }
714 /**
715 **          None
716 */
717 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
718 {
719     EmdHandle::getBindingHandle(ID);
720 }
721 /**
722 **          None
723 */
724 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
725 {
726     EmdHandle::getShellHandle(id);
727 }
728 /**
729 **          None
730 */
731 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
732 {
733     EmdHandle::setSessionID(id);
734 }
735 /**
736 **          None
737 */
738 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
739 {
740     EmdHandle::getBindingHandle(ID);
741 }
742 /**
743 **          None
744 */
745 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
746 {
747     EmdHandle::getShellHandle(id);
748 }
749 /**
750 **          None
751 */
752 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
753 {
754     EmdHandle::setSessionID(id);
755 }
756 /**
757 **          None
758 */
759 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
760 {
761     EmdHandle::getBindingHandle(ID);
762 }
763 /**
764 **          None
765 */
766 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
767 {
768     EmdHandle::getShellHandle(id);
769 }
770 /**
771 **          None
772 */
773 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
774 {
775     EmdHandle::setSessionID(id);
776 }
777 /**
778 **          None
779 */
780 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
781 {
782     EmdHandle::getBindingHandle(ID);
783 }
784 /**
785 **          None
786 */
787 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
788 {
789     EmdHandle::getShellHandle(id);
790 }
791 /**
792 **          None
793 */
794 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
795 {
796     EmdHandle::setSessionID(id);
797 }
798 /**
799 **          None
800 */
801 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
802 {
803     EmdHandle::getBindingHandle(ID);
804 }
805 /**
806 **          None
807 */
808 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
809 {
810     EmdHandle::getShellHandle(id);
811 }
812 /**
813 **          None
814 */
815 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
816 {
817     EmdHandle::setSessionID(id);
818 }
819 /**
820 **          None
821 */
822 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
823 {
824     EmdHandle::getBindingHandle(ID);
825 }
826 /**
827 **          None
828 */
829 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
830 {
831     EmdHandle::getShellHandle(id);
832 }
833 /**
834 **          None
835 */
836 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
837 {
838     EmdHandle::setSessionID(id);
839 }
840 /**
841 **          None
842 */
843 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
844 {
845     EmdHandle::getBindingHandle(ID);
846 }
847 /**
848 **          None
849 */
850 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
851 {
852     EmdHandle::getShellHandle(id);
853 }
854 /**
855 **          None
856 */
857 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
858 {
859     EmdHandle::setSessionID(id);
860 }
861 /**
862 **          None
863 */
864 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
865 {
866     EmdHandle::getBindingHandle(ID);
867 }
868 /**
869 **          None
870 */
871 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
872 {
873     EmdHandle::getShellHandle(id);
874 }
875 /**
876 **          None
877 */
878 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
879 {
880     EmdHandle::setSessionID(id);
881 }
882 /**
883 **          None
884 */
885 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
886 {
887     EmdHandle::getBindingHandle(ID);
888 }
889 /**
890 **          None
891 */
892 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
893 {
894     EmdHandle::getShellHandle(id);
895 }
896 /**
897 **          None
898 */
899 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
900 {
901     EmdHandle::setSessionID(id);
902 }
903 /**
904 **          None
905 */
906 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
907 {
908     EmdHandle::getBindingHandle(ID);
909 }
910 /**
911 **          None
912 */
913 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
914 {
915     EmdHandle::getShellHandle(id);
916 }
917 /**
918 **          None
919 */
920 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
921 {
922     EmdHandle::setSessionID(id);
923 }
924 /**
925 **          None
926 */
927 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
928 {
929     EmdHandle::getBindingHandle(ID);
930 }
931 /**
932 **          None
933 */
934 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
935 {
936     EmdHandle::getShellHandle(id);
937 }
938 /**
939 **          None
940 */
941 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
942 {
943     EmdHandle::setSessionID(id);
944 }
945 /**
946 **          None
947 */
948 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
949 {
950     EmdHandle::getBindingHandle(ID);
951 }
952 /**
953 **          None
954 */
955 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
956 {
957     EmdHandle::getShellHandle(id);
958 }
959 /**
960 **          None
961 */
962 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
963 {
964     EmdHandle::setSessionID(id);
965 }
966 /**
967 **          None
968 */
969 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
970 {
971     EmdHandle::getBindingHandle(ID);
972 }
973 /**
974 **          None
975 */
976 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
977 {
978     EmdHandle::getShellHandle(id);
979 }
980 /**
981 **          None
982 */
983 void EmdHandle::setSessionID(EMD_CLIENT_SESSION_ID id)
984 {
985     EmdHandle::setSessionID(id);
986 }
987 /**
988 **          None
989 */
990 void EmdHandle::getBindingHandle(EMD_CLIENT_SESSION_ID ID)
991 {
992     EmdHandle::getBindingHandle(ID);
993 }
994 /**
995 **          None
996 */
997 void EmdHandle::getShellHandle(EMD_CLIENT_SESSION_ID id)
998 {
999     EmdHandle::getShellHandle(id);
1000 }

```

```

479     ** Routine: getStdoutPipe
480     ** Inputs: None
481     ** Outputs: None
482
483     ** Return Codes: int stdouted - the stdout descriptor of the service
484
485     ** Purpose: Returns the stdout handle of the service.
486
487
488     ** Inputs: None
489     ** Outputs: None
490
491     ****
492
493     int
494     EDMDHandle::getStdoutPipe()
495     {
496         return stdoutPipes;
497     }
498
499     ****
500     ** Routine: setStdoutPipe
501     ** Inputs: int handle - the stdout handle of the private service
502
503     ** Outputs: None
504
505     ** Purpose: Sets the stdout handle of the private service.
506
507     ** Return Codes: None
508
509     ** Inputs: None
510     ** Outputs: None
511
512     ****
513
514     **
515     void
516     EDMDHandle::setStdoutPipe(int handle)
517     {
518         stdoutPipe = handle;
519     }
520
521     ****
522
523     ** Routine: getStderrPipe
524     ** Inputs: None
525     ** Outputs: None
526
527     ** Return Codes: int stderrPipe - the stderr descriptor of the service
528
529     ** Purpose: Returns the stderr descriptor of the service.
530
531
532     ** Inputs: None
533     ** Outputs: None
534
535     ** Purpose: Returns the stderr descriptor of the service.
536
537
538     ** Inputs: None
539     ** Outputs: None
540
541     ****
542
543     int
544     EDMDHandle::getStderrPipe()
545     {
546         return stderrPipe;
547     }
548
549     ** Purpose: Sets the stderr handle of the service.
550
551     ** Return Codes: None
552
553     ** Inputs: int handle - the stderr handle of the service
554     ** Outputs: None
555
556     ****
557
558     void
559     EDMDHandle::setStderrPipe(int handle)
560     {
561         stderrPipe = handle;
562     }
563
564     ****

```


	Function Index	Page 1
GetSCHandle	12 (EMONDHANDLengPapi.cci)	
GetSMHandle	13 (EMONDHANDLengPapi.cci)	
LockHandleEx	3 (EMONDHANDLengPapi.cci)	
LookupHandleEx	7 (EMONDHANDLengPapi.cci)	
UnlockHandleEx	4 (EMONDHANDLengPapi.cci)	
ReleaseHandleEx	14 (EMONDHANDLengPapi.cci)	
GetHandleSet	11 (EMONDHANDLengPapi.cci)	
GetCharacterSet	6 (EMONDHANDLengPapi.cci)	
GetFontSet	5 (EMONDHANDLengPapi.cci)	
GetFileSet	2 (EMONDHANDLengPapi.cci)	
GetIconSet	9 (EMONDHANDLengPapi.cci)	


```

1  /* Copyright: 1996,1997 EMC Corporation
2  */
3  /* $Header: /EDMDHandleMgtApI.cc,v 1.1 1997/02/05 20:49:15 $ */
4
5  // Mission Statement: An API to manage the handle sets/objects
6  // Primary Data acted on:
7  // Compile-Time Options:
8
9  * Basic idea here: This API manages the handle sets. Multiple threads
10 * need access to the handles to do IO.
11 * Each time an fd_set is modified or used we lock
12 * a mutex to make sure access is serialized.
13 *
14 */
15
16 #include <sys/types.h>
17 #include <sys/conf.h>
18 #include <sys/time.h>
19 #include <sys/param.h>
20
21 #ifndef _definedInLinc
22 static char RCS_Id [] = "@(#) $RCSfile: EDMDHandleMgtApI.cc,v $"
23 RCS_Id[44] = '0';
24 RCS_Id[45] = ' ';
25 RCS_Id[46] = '"';
26 RCS_Id[47] = '1';
27 RCS_Id[48] = '9';
28 RCS_Id[49] = '7';
29 RCS_Id[50] = '0';
30 RCS_Id[51] = '9';
31 RCS_Id[52] = '8';
32 RCS_Id[53] = '7';
33 RCS_Id[54] = '6';
34 RCS_Id[55] = '5';
35 RCS_Id[56] = '4';
36 RCS_Id[57] = '3';
37 RCS_Id[58] = '2';
38 RCS_Id[59] = '1';
39 RCS_Id[60] = '0';
40 RCS_Id[61] = '"';
41 RCS_Id[62] = ' ';
42 RCS_Id[63] = '1';
43 RCS_Id[64] = '0';
44 RCS_Id[65] = '9';
45 RCS_Id[66] = '8';
46 RCS_Id[67] = '7';
47 RCS_Id[68] = '6';
48 RCS_Id[69] = '5';
49 RCS_Id[70] = '4';
50 RCS_Id[71] = '3';
51 RCS_Id[72] = '2';
52 RCS_Id[73] = '1';
53 RCS_Id[74] = '0';
54 RCS_Id[75] = '"';
55
56 // These are values for the highest number handle that is part of a
57 // given set. Keep in mind that I has to be added to this number to
58 // select on the highest handle used.
59
60 int highestDevr = 0;
61
62 highStDevr = 0;
63
64 static pthread_mutex_t g_fdMutex = PTHREAD_MUTEX_INITIALIZER;
65 static pthrd_mutex_t g_handleMutex = PTHRD_MUTEX_INITIALIZER;

```

```

66 /*********************************************************************
67 * * * * * Routine: initFDSets
68 * * * * * Inputs: None
69 * * * * * Outputs: None
70 * * * * * Returns: None
71 * * * * * Return Codes: None
72 * * * * * Purpose: Initialize fd sets and mutex.
73 * * * * * ****
74 * * * * * ****
75 * * * * * ****
76 * * * * * ****
77 * * * * * ****
78 * * * * * ****
79 */
80
81 static void
82 initFDsets()
83 {
84     FD_ZERO(&fdSetModified);
85     FD_ZERO(&stcHandleModifed);
86     FD_ZERO(&stcHandleModifed);
87     FD_ZERO(&stcHandleModifed);
88 }
89
90 pthread_mutex_init(&g_fdSetMutex, NULL);
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
815
816
817
817
818
819
819
820
821
822
823
823
824
825
825
826
827
827
828
829
829
830
831
831
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
1671
1672
1672
1673
1673
1674
```

```

93 //*****
94 /**
95 *  * Routine: LockHandleMutex
96 *  * Inputs: None
97 *  * Outputs: None
98 *  * Return Codes: None
99 */
100 /**
101 *  * Returns: None
102 */
103 /**
104 *  * Purpose: Lock the mutex for the handle tree object
105 */
106 /**
107 */
108 /**
109 static void
110 LockHandleMutex()
111 {
112     static boolean bFirst = TRUE;
113     if (bFirst == TRUE)
114     {
115         bFirst = FALSE;
116         pthread_mutex_init(&g_HandleTreeMutex, NULL);
117     }
118 }
119 */
120 /**
121 */
122 pthread_mutex_lock(&g_HandleTreeMutex);
123 */

```

```

123 //*****
124 /**
125 *  * Routine: UnlockHandleMutex
126 *  * Inputs: None
127 *  * Outputs: None
128 *  * Return Codes: None
129 */
130 /**
131 *  * Returns: None
132 */
133 /**
134 *  * Purpose: Unlock the mutex for the handle tree object
135 */
136 /**
137 */
138 /**
139 static void
140 unlockHandleMutex()
141 {
142     pthread_mutex_unlock(&g_HandleTreeMutex);
143 }

```

```

145 //*****
146 ** Routine: getStdoutSet
147 ** Inputs: None
148 ** Outputs: None
149 */
150 int
151 {
152     /* Return Codes:
153      * Ed_Set * - the stdoutSet...
154      * Ed_Set * - the stdoutSet after the stdoutSetModified was
155      * Purpose: Returns the stdoutSet Ed_Set after the stdoutSetModified was
156      * copied into it. Modified is the most recent copy.
157      * Purpose: Returns the stderrSet Ed_Set after the stderrSetModified was
158      * copied into it. Modified is the most recent copy.
159      */
160
161 //*****
162 int
163 getStdoutSet(Ed_Set *yourset, int *highHandle, int *status)
164 {
165     if (*status == NULL)
166     {
167         return -1;
168     }
169     if (*yourset == NULL || *highHandle == NULL)
170     {
171         *status = HANDLEMGR_BAD_PARAM;
172         return -1;
173     }
174     pthread_mutex_lock(&fd_setMutex);
175     stdOutSet = stdOutSetModified;
176     memcpy(yourset, &stdOutSet, sizeof(fd_set));
177     *highHandle = highHandle;
178     pthread_mutex_unlock(&fd_setMutex);
179     return 0;
180 }
181
182 //*****
183 int
184 getStderrSet(Ed_Set *yourset, int *highHandle, int *status)
185 {
186     if (*status == NULL || *highHandle == NULL)
187     {
188         if (*yourset == NULL || *highHandle == NULL)
189         {
190             *status = HANDLEMGR_BAD_PARAM;
191             return -1;
192         }
193         *highHandle = highHandle;
194         pthread_mutex_lock(&fd_setMutex);
195         stdErrSet = stdErrSetModified;
196         memcpy(yourset, &stdErrSet, sizeof(fd_set));
197         *highHandle = highHandle;
198         pthread_mutex_unlock(&fd_setMutex);
199         return 0;
200     }
201 }

```

```

216 ****
217 /**
218 ** Routine: LookupHandleSet
219 ** Inputs: DD_CLIENT_SESSION_ID *sess - the session ID to lookup with
220 **          int *status - a place to put a status if something goes
221 **          wrong.
222 ** Outputs: int *status - a place to put a status if something goes
223 **          wrong.
224 **          EMDHANDLE **hs - the handle set to return
225 **          EMDHANDLE *ret - 0 for success and non-zero otherwise
226 **          Purpose: Looks up a handle set.
227 **          ****
228 **          ****
229 */
230
231 int
232 LookupHandleSet(DD_CLIENT_SESSION_ID *sess, EMDHANDLE **hs, int *status)
233 {
234     EMDHANDLE *ret, *handleobj;
235
236     if (status == NULL)
237         return -1;
238
239     if (sess == NULL)
240         return -1;
241
242     if (hs == NULL || sess == NULL)
243     {
244         *status = HANDLEMGR_BAD_PARAM;
245         return -1;
246     }
247
248     handleobj = new EMDHandle();
249
250     if (handleobj == NULL)
251     {
252         *status = HANDLEMGR_NO_MEMORY;
253         return -1;
254     }
255
256     handleobj->setSessionID(*sess);
257
258     LockHandleMutex();
259
260     ret = (EMDHandle *) _handleTree.find(handleobj);
261
262     UnlockHandleMutex();
263
264     delete handleobj;
265
266     if (ret == NULL)
267     {
268         *status = HANDLEMGR_LOOKUP_FAILED;
269         return -1;
270     }
271
272     *hs = ret;
273
274     return 0;
275 }

```

```

298 ****
299 */
300 */
301 * * Routine: newHandleSet
302 * * Inputs: int stdouthandle - the handle to send commands to auxrc
303 * *          int stderrhandle - the handle to receive responses from auxrc
304 * *          auxrc
305 * *          ITC stderrhandle_t *comhandle - the connection handle
306 * *          ELinkShellObject_t *shell - the shell handle
307 */
308 */
309 * * Outputs: int *status - a place to put a status if something goes
310 * *          wrong.
311 * *          Return Codes:
312 * *          0 for success and non-zero otherwise
313 * *          Purpose: Creates a new handle set.
314 */
315 ****
316 */
317 int
318 newHandleSet(
319     _EDMDClientSessionId *sess, int stdouthandle, int stderrhandle,
320     _EDMDHandle comhandle, _EDMDHandle shell,
321     _EDMDHandle *ret);
322 */
323 int
324 _EDMDHandle stderr;
325 static int
326 _EDMDHandle stdout;
327 */
328 int
329 {
330     int *status;
331     _EDMDHandle *ret;
332     int flags = 0;
333     static boolean _EDMDHandle first = TRUE;
334     if (first == TRUE)
335     {
336         if (interpret(1, first = FALSE));
337         if (status == NULL)
338         {
339             if (*status = _EDMDHandle_BAD_PARAM);
340             return -1;
341         }
342     }
343     return -1;
344 }
345 */
346 */
347 if (handle == NULL)
348 {
349     *status = _EDMDHandle_NO_MEMORY;
350     return -1;
351 }
352 handle->setStdoutPipe(stdouthandle);
353 handle->setStderrPipe(stderrhandle);
354 handle->setBindingHandle(comhandle);
355 handle->setShellHandle(shell);
356 handle->setSessionID(*sess);

```

```

337 /*********************************************************************
338 */
339 /**
340 * Routine: getHandleSet
341 *
342 * Inputs: DD_CLIENT_SESSION_ID *sess - a session ID to use to look
343 *          up the handle set
344 */
345 /**
346 * Outputs: int *status - a place to put a status if something goes
347 *          wrong.
348 *          int *sout - stdout descriptor for the service
349 *          int *serr - stderr descriptor for the service
350 */
351 /**
352 * Return Codes:
353 *    0 for success and non-zero otherwise
354 */
355 /**
356 * Purpose: Removes a handle set.
357 */
358 /**
359 * ****
360 */
361 /**
362 * ****
363 */
364 /**
365 * int
366 * getHandleSet( IN DD_CLIENT_SESSION_ID *sess, OUT int *sout, OUT int *serr,
367 *               OUT int *status)
368 */
369 {
370     /**
371      * REMOVEDHANDLE *handle;
372      * int;
373      * if (sstatus == NULL)
374      * {
375      *     return -1;
376      * }
377      * if (sess == NULL || sout == NULL || serr == NULL)
378      * {
379      *     *status = HANDLNGR_BAD_PARAM;
380      *     return -1;
381      * }
382      */
383
384     lret = LookUpHandleSet(sess, handle, status);
385
386     if (lret != 0)
387     {
388         return lret;
389     }
390     else if (handle == NULL)
391     {
392         *sout = handle->getStdoutPipe();
393         *serr = handle->getStderrPipe();
394     }
395
396     return 0;
397 }
398 */
399 /**
400 * Routine: GetCSCHandle
401 */
402 /**
403 * Inputs: DD_CLIENT_SESSION_ID *sess - a session ID to use to look
404 *          up the handle set
405 */
406 /**
407 * Outputs: int *status - a place to put a status if something goes
408 *          wrong.
409 */
410 /**
411 * RPC-Binding handle_t *cscb - binding handle for this session
412 */
413 /**
414 */
415 /**
416 * int
417 * getCSCHandle( IN DD_CLIENT_SESSION_ID *sess, OUT rpc_binding_handle_t **cscb,
418 *                OUT int *status)
419 */
420 {
421     /**
422      * REMOVEDHANDLE *handle;
423      * int;
424      * if (sstatus == NULL)
425      * {
426      *     return -1;
427      * }
428      * if (sess == NULL || cscb == NULL || status == NULL)
429      * {
430      *     if (lret != 0)
431      *     {
432      *         return lret;
433      *     }
434      *     else if (handle == NULL)
435      *     {
436      *         *cscb = handle->getBindingHandle();
437      *     }
438      *     else if (handle == NULL)
439      *     {
440      *         return -1;
441      *     }
442      *     else if (handle->getStdoutPipe() == NULL ||
443      *             handle->getStderrPipe() == NULL)
444      *     {
445      *         return -1;
446      *     }
447     */
448 }
449 */

```

```

489 // ****
490 // ** Routine: GetShellHandle
491 // ** Inputs: DB_client_session_id *sess - a session ID to use to look
492 //           up the handle set
493 // ** Outputs: int *status - a place to put a status if something goes
494 //           wrong.
495 //           int *status - a place to put a status if something goes
496 //           wrong.
497 // ** Purpose: Returns shell handle for this session
498 //           Return Codes:
499 //           0 for success and non-zero otherwise
500 //           Purpose: Deletes a handle set.
501 //           Purpose: Removes a handle set.
502 //           Purpose: Returns shell handle.
503 // ****
504 // ****
505 // ****
506 // ****
507 int GetShellHandle( int _client_session_id *sess, OUT ElinkShellObjPtr_t *shell,
508 IN DO_CLIENT_SESSION_ID *sess, OUT ElinkHandleObjPtr_t *hand,
509 I
510 {
511     ElinkHandle *handle;
512     int ret;
513     if (sess == NULL || shell == NULL || status == NULL)
514     {
515         return -1;
516     }
517     handle = lookupHandleSet(sess, &handle, status);
518     if (ret == 0)
519     {
520         return ret;
521     }
522     else if (handle == NULL)
523     {
524         return -1;
525     }
526     shell = handle->getShellHandle();
527     if (ret == 0)
528     {
529         return 0;
530     }
531     return ret;
532 }
533 else if (handle == NULL)
534 {
535     return -1;
536 }
537 }
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829

```

Page 15 of 16 deleteHandleSet Thu Dec 27 11:38:18 2007

Page 16 of 16

```
Page 15 of 16
```

Line	Code
592 1	FD_CLOSETIME -> getSetCloseTime(); &asocSetModified);
593 1	FD_CURRENTR -> getSetCurrentTime(); &asocSetModified);
595 1	pthread_mutex_unlock(&g_fSetHtx);
597 1	bh = ret -> getBindingHandle();
599 1	case_FREE_BINDING(bh, 0, &err);
601 1	shell = ret -> getShellHandle();
603 1	ERLinkDestroyObj(hand, *shell);
605 1	delete ret;
607 1	return 0;
608 1	

EDWARD HARRIS MARVEL on 15
TUE DEC 27 1:38:18 2007

EDMDHandleMgApi.cc 15

FHU Dec 2 / 11:38:18 2007

ԵՐԱՎԵՐ: ԹԻ: ԽԱԿՈՎԻ ՏՐԱՋԵ

卷之三